

ORACLE7 I/O Monitoring and Tuning

Andrew Rivenes

Neil Jensen

Lawrence Livermore National Laboratory

Introduction

This paper will outline the techniques that we have used to identify and track the amount of file input/output (I/O) that takes place in our Oracle7 databases. We will show how to setup a monitoring process that can capture the file I/O taking place in the database at regular intervals. We have included the tables that we use to store this information and the reports that can be used to view this information.

We will then explore how we have been able to use this information to identify processes within the database that perform large amounts of file I/O. We will cover our experiences with both an Oracle Applications and with in-house written applications. We will then show the successes that we have had in the reduction of overall file I/O. We will give a short description of the techniques that we use in tuning individual SQL statements including being able to dynamically identify the SQL statements associated with a running process.

The Purpose

We believe that Oracle7 I/O monitoring and tuning consists of more than just balancing I/O evenly across spindles or creating striped tablespaces, and more to the point that Oracle7 I/O tuning means REDUCING I/O. Time and time again we see queries that take hours reduced to running in minutes by a simple change in the SQL code or the addition

of an index to a table. The key is identifying the queries that cause excessive I/O, and this is the purpose for the monitoring system that we are describing.

In order to accomplish this we decided that a monitoring system was needed. This could give us both a point in time snapshot of how hard the database was working and more importantly, the ability to historically track the performance of the database. The historical data gives us some metrics to use when upgrading, both hardware and software, and it helps us track when performance degrades.

The Environment

Our environment at Lawrence Livermore National Laboratory (the Lab) consists of running Oracle databases for the Administrative Information Systems (AIS) department. AIS performs the administrative computing for the Lab (i.e. Accounts Payable, General Ledger, etc). In that capacity we manage several Oracle databases all running on Hewlett Packard Unix (HP-UX) machines. These machines are dedicated to running Oracle databases exclusively. We also manage several different client applications connecting to these databases. These applications run on an assortment of Sun, DEC VAX, HP-UX, Macintosh and Windows machines.

This includes our Oracle Applications environment where the database is running on a HP9000 Series 800 Model T500 (4 cpu's).

There is 760MB of memory and the database is configured with a 125MB SGA. The current database version is Oracle 7.1.4.1.10. The operating system is HP-UX 9.04E. Oracle Applications requires the OPTIMIZER_MODE init.ora parameter to be set to "RULE". The database size is about 19GB. The application code is running on an HP9000 Series 755 machine running Release 10.4.2. We are using Oracle's General Ledger, Accounts Payable, Purchasing and some in-house applications.

The Monitor

When we started tuning databases we accessed statistics from Oracle's V\$ views as well as statistics from the operating system. It soon became apparent that we needed a way to track trends in this data.

The monitor routine is basically one scheduled script that records database and operating system information on a regular interval. We chose to run this job hourly through cron but the interval is really dependent on the data needs of the organization. The monitor routine is comprised of one cron entry and a Korn shell script on the controlling system, and a Korn shell and SQL script on each of the database server machines.

All of the SQL and UNIX scripts are in Appendix A.

The Data

The data consists of a set of tables owned by an Oracle userid on each of the monitored databases. This could be the SYSTEM account, but we prefer to keep our custom data separate from Oracle's tables and have setup an account for this purpose. This Oracle userid requires some system privileges for the

monitor script to run and it will need some system privileges in order to create the needed database objects. The privileges are listed in Appendix A.

Formatting The Data

We have written SQL*Plus reports that provide us with a daily summary of the previous day's activities. Of particular interest is the I/O by data file. This information lets us know at a glance whether there has been abnormally high I/O activity. Based on this information we can further break down a data file's I/O hourly to see approximately when the activity occurred. We provided the view we use to check hourly I/O in Appendix A. It may have been one or more spikes, or it may have been constant for an entire day.

An example of the daily summary report is in Appendix B.

Other useful information would be the total number of users, peak number of users, SGA hit ratio and total I/O by disk drive.

Interpreting The Data

When we started we made the assumption that we had an I/O problem, or that at the least, we could reduce the amount of I/O we were performing. Now that we had data and a way to look at it we needed to figure out what it meant. We first started with the data file with the highest I/O counts and proceeded to break down that data file's I/O by hour. See Figure 1.

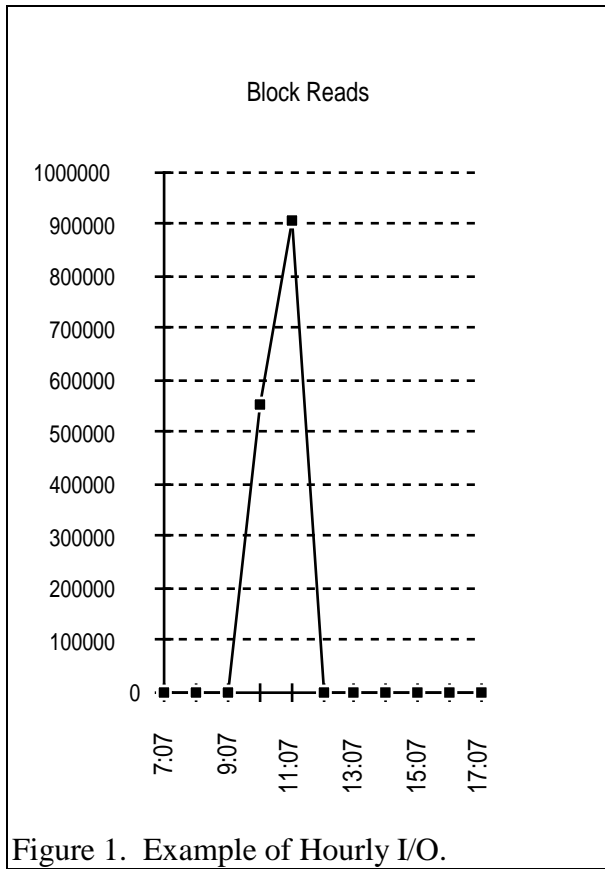


Figure 1. Example of Hourly I/O.

This information identifies the periods of time with high I/O rates. We then went back to the concurrent manager tables, specifically the FND_CONCURRENT_REQUESTS table, and listed all of the concurrent jobs that ran during the period of time in question. We then narrowed these jobs down to the jobs which were most likely to be accessing data in the target data file.

In general there were just one or two offending jobs. Once a job or process was identified it was just a matter of determining the SQL being executed and tuning it. In our case it was mostly report jobs which made finding the SQL easy. For Oracle reports we worked with the Application Developer and normally were able to add or modify an index to dramatically reduce the I/O for that report. If the report was one written here, again we would work with the Application Developer

and either work on the indexes or the structure of the SELECT statement.

Sometimes it was obvious what column needed to be indexed. Other times we would run the report several times in development either modifying the SELECT statement or changing the indexes with SQL_TRACE turned on.

One thing you need to be aware of is the impact of adding an index. You may make one report run twice as fast, but cause 10 others to take much longer. It is this type of behavior that makes the process ongoing.

When we could not identify where the long running code was coming from we would use the view V\$SQLAREA and determine which SQL statement was causing the I/O. This is a SELECT statement that can be used to identify an active session's SQL (input is the session's session id):

```
SELECT a.sid, a.username, c.sql_text
FROM v$session a, v$open_cursor b,
     v$sqlarea c
WHERE a.sid = &<active_session_id>
AND a.saddr = b.saddr
AND b.ADDRESS = c.ADDRESS
AND b.HASH_VALUE = c.HASH_VALUE;
```

Results

We have successfully used this process to reduce hours of processing from our daily reporting process. There was a set of daily reports that started at 3:00 am and would finish about 11:00 am. Using the monitoring tool to identify the problem reports, we were able to shorten the process by 3 hours.

There are many other additional benefits from capturing the data. You can use the statistics to balance your I/O over the available disk drives and controllers. You can use the user count information to see growth patterns in

the system. You can correlate changes in I/O activity with the release of new code and/or Oracle Applications upgrades.

basis and can act proactively rather than reactively.

Conclusion

In conclusion, we have shown how a fairly simple monitoring system can provide a wealth of information. By accumulating daily database statistics into a historical record a better picture can be obtained of the impact of changes to the environment. Whether it be adding more CPU's, striping tablespaces across spindles, adding more users, or adding new applications, you can have hard data to document the effect rather than a seat of the pants observation. We believe that no matter how much you tune the database parameters, spread I/O, or add hardware, the real performance increases are obtained by tuning the applications. This system provides a mechanism to do just that, and perhaps most importantly, you will have a handle on what is going on in your database on a day to day

About the Authors

Andy Rivenes became an Oracle DBA in 1992. Prior to that he was a Systems Programmer working mostly with IBM mainframes.

Neil Jensen has been an Oracle DBA since 1989. This is also when he began working with Oracle Applications. Prior to this Neil worked with network databases and was an Application Developer.

Both Andy and Neil work in the Data Management Team at Lawrence Livermore National Laboratory. If you want to speak with either of them, look for the guys in shorts and tee-shirts.

+++++

Appendix A: The SQL scripts and Unix Korn shell scripts.

The first two scripts should be run once to create the collection tables and create the database procedure that will be used to populate the data. The third script is a view used to show the hourly file I/O.

In order to run these scripts, the Oracle account will need these privileges:

```
SELECT ANY TABLE
CREATE TABLE
CREATE VIEW
CREATE SEQUENCE
CREATE PROCEDURE
```

SQL Script #1: Build collection tables:

```
/*
FILE:          dmtmon_install.sql
DESCRIPTION:
  This script installs the objects necessary.

AUTHOR:  Andy Rivenes & Neil Jensen
DATE:    10/27/94  Rev: 6/7/95
*/

DROP SEQUENCE stats_id_no_seq;
CREATE SEQUENCE stats_id_no_seq
  INCREMENT BY 1
  START WITH 1
  NOCACHE;

DROP TABLE db_level_stats;
CREATE TABLE db_level_stats
  (stats_id_no          NUMBER
   constraint db_level_stats_nn1 NOT NULL
   run_time_date       DATE
   constraint db_level_stats_nn2 NOT NULL
   sid_name            VARCHAR2(9)
   constraint db_level_stats_nn3 NOT NULL,
   node_name           VARCHAR2(6),
   oracle_database_version_desc VARCHAR2(64),
   active_total_user_cnt_qty NUMBER,
   active_distinct_user_cnt_qty NUMBER,
   redo_log_no         NUMBER,
   logical_disk_reads_qty NUMBER,
   physical_disk_reads_qty NUMBER)
  STORAGE (INITIAL 64K NEXT 64K PCTINCREASE 0);

DROP TABLE db_file_level_stats;
CREATE TABLE db_file_level_stats
  (stats_id_no          NUMBER
   constraint db_file_level_stats_nn1 NOT NULL,
   oracle_file_id_no    NUMBER
   constraint db_file_level_stats_nn2 NOT NULL,
   physical_reads_qty   NUMBER,
   physical_writes_qty  NUMBER,
   physical_block_reads_qty NUMBER,
   physical_block_writes_qty NUMBER )
  STORAGE (INITIAL 128K NEXT 64K PCTINCREASE 0);

DROP TABLE node_level_stats;
CREATE TABLE node_level_stats
  (stats_id_no          NUMBER
   constraint node_level_stats_nn1 NOT NULL,
   cpu_pct_user         NUMBER,
   cpu_pct_system       NUMBER,
   cpu_pct_idle         NUMBER,
   load_avg_qty_1_min   NUMBER,
   load_avg_qty_5_min   NUMBER,
   load_avg_qty_15_min  NUMBER)
  STORAGE (INITIAL 128K NEXT 64K PCTINCREASE 0);
```

SQL Script #2: Create the database procedure:

```
create or replace procedure dmtmon_collect_stats
( stats_id_no IN OUT NUMBER )
as
/*
FILE: dmtmon_collect_stats_proc.sql
DESCRIPTION:
  PLSQL script to monitor an Oracle database. This script inserts
  into a table a snapshot of current database stats and is meant
  to be run on a periodic basis in order to gather historical
  monitoring data.

AUTHOR: Andy Rivenes & Neil Jensen
DATE: 06/15/95

REQUIREMENTS:
  This script requires a sequence called stats_id_no_seq and the
  following tables:

  db_level_stats
  db_file_level_stats

  Since the data stored is point in time a database shutdown
  can affect the numbers.
*/

CURSOR filestat_cursor IS
  SELECT file#, phydrds, phywrts, phyblkrd, phyblkwrt
  FROM v$filestat
  ORDER BY file#;

filestat_record filestat_cursor%ROWTYPE;

sid_name          VARCHAR2(9);
node_name         VARCHAR2(6);
oracle_database_version_desc VARCHAR2(64);
active_total_user_cnt_qty NUMBER;
active_distinct_user_cnt_qty NUMBER;
redo_log_no       NUMBER;
consistent_gets   NUMBER;
db_block_gets     NUMBER;
logical_disk_reads_qty NUMBER;
physical_disk_reads_qty NUMBER;

BEGIN
/* Collect database level statistics and store in
db_level_stats table */

SELECT stats_id_no_seq.nextval INTO stats_id_no
FROM SYS.DUAL;

SELECT name INTO sid_name
FROM v$database;

SELECT substr(program,8,6) INTO node_name
FROM v$process
```

```

WHERE program LIKE '%(PMON)%';

SELECT banner INTO oracle_database_version_desc
FROM v$version
WHERE UPPER(banner) LIKE 'ORACLE%';

SELECT value INTO active_total_user_cnt_qty
FROM v$sysstat
WHERE statistic# = 1
AND class = 1;

SELECT COUNT(DISTINCT(username)) into active_distinct_user_cnt_qty
FROM v$session;

SELECT sequence# INTO redo_log_no
FROM v$log
WHERE status = 'CURRENT';

SELECT value INTO consistent_gets
FROM v$sysstat
WHERE statistic# = 38
AND class = 8;
SELECT value INTO db_block_gets
FROM v$sysstat
WHERE statistic# = 37
AND class = 8;
logical_disk_reads_qty := consistent_gets + db_block_gets;

SELECT value INTO physical_disk_reads_qty
FROM v$sysstat
WHERE statistic# = 39
AND class = 8;

INSERT INTO db_level_stats
VALUES(stats_id_no, SYSDATE, sid_name, node_name,
       oracle_database_version_desc,
       active_total_user_cnt_qty, active_distinct_user_cnt_qty,
       redo_log_no, logical_disk_reads_qty, physical_disk_reads_qty);

/* Load the file based statistics into db_file_level_stats table */

OPEN filestat_cursor;
<<L1>> LOOP
  FETCH filestat_cursor INTO filestat_record;
  EXIT L1 WHEN filestat_cursor%NOTFOUND;

  INSERT INTO db_file_level_stats
  VALUES(stats_id_no, filestat_record.file#,
         filestat_record.phyrds, filestat_record.phywrts,
         filestat_record.phyblkrd, filestat_record.phyblkwrt);

END LOOP L1;
CLOSE filestat_cursor;
END;
.
/

```

This is the view that will allow you to see hourly increments in file I/O.

```
CREATE OR REPLACE VIEW stats_file_io_v2 (  
    run_time_date,  
    sid_name,  
    oracle_file_id_no,  
    physical_block_reads_qty,  
    physical_block_writes_qty )  
AS SELECT  
    SUBSTR(TO_CHAR(c.run_time_date, 'DD-MON-YY HH24:MM'),1,15) run_time_date,  
    c.sid_name "sid_name",  
    a.oracle_file_id_no  
    a.physical_block_reads_qty - b.physical_block_reads_qty physical_block_reads_qty,  
    a.physical_block_writes_qty - b.physical_block_writes_qty physical_block_writes_qty      FROM  
db_file_level_stats a, db_file_level_stats b, db_level_stats c  
WHERE a.stats_id_no-1 = b.stats_id_no  
AND    a.stats_id_no = c.stats_id_no  
AND    a.oracle_file_id_no = b.oracle_file_id_no ;
```

The next three scripts are used to run the collection on a periodic basis. We use cron from a central machine to run the collection each hour on all of our database machines.

The cron entry to start the collections every hour:

```
00 * * * * <directory>/run_remote_data_collections > /dev/null 2>&1
```

This Korn shell on the central machine that runs the remote data collection:

```
#!/bin/ksh  
#  
# Monitor Database Information  
#  
. <environment setup script>  
rsh -l <unix id> -n <db machine1> <directory>/dmtmon_collect_stats  
rsh -l <unix id> -n <db machine2> <directory>/dmtmon_collect_stats  
exit
```

This Korn shell script must be put on each database machine:

```
#!/bin/ksh  
#  
# This script sets the environment variable cpu to the current  
# cpu percent for user as returned from vmstat and sets cur_load  
# to the current 15 minute load avg as returned from uptime.  
# NOTE: This script is written specific to HP-UX  
#  
if [ -f tmp.lst ]  
then  
    rm tmp.lst  
fi  
vmstat 5 2 > tmp.lst  
set -A t_array `tail -n 1 tmp.lst`
```

```

ucpu='echo ${t_array[15]}'
scpu='echo ${t_array[16]}'
icpu='echo ${t_array[17]}'
rm tmp.lst
load1='uptime | awk -F, '{print $4}' - | sed 's;load average::;'
load2='uptime | awk -F, '{print $5}' -'
load3='uptime | awk -F, '{print $6}' -'
./oracle/script/ora7setup
sqlplus <oracle_id>/<passwd> << STOP
VARIABLE stats_id_no NUMBER;
EXECUTE dmtmon_collect_stats(:stats_id_no);
INSERT INTO node_level_stats
  VALUES(:stats_id_no,${ucpu},${scpu},${icpu},${load1},${load2},${load3});
COMMIT;
EXIT;
STOP
#
exit

```

Reporting SQL scripts.

This next script is used to print out a summary report from the previous days data collection.

```

SET LINESIZE 80;
SET PAGESIZE 66;
SET VERIFY OFF;
SET FEEDBACK OFF;

COLUMN stats_id_no_max HEADING 'Stats ID|Max' FORMAT 9990 NEW_VALUE stats_id_n
o_max;
COLUMN stats_id_no_min HEADING 'Stats ID|Min' FORMAT 9990 NEW_VALUE stats_id_n
o_min;

SELECT MAX(a.stats_id_no) stats_id_no_max, MIN(a.stats_id_no) stats_id_no_min
  FROM db_level_stats a
 WHERE TO_CHAR(a.run_time_date,'DD-MON-YY') =
        TO_CHAR(SYSDATE-1,'DD-MON-YY')
 AND   to_char(run_time_date,'HH24') > 8
 AND   to_char(run_time_date,'HH24') < 25;

spool dmtmon_print.lst;
PROMPT "Database Performance Report";

SET HEAD on;

COLUMN lreads HEADING 'Logical|Reads' FORMAT 9999,999,999;
COLUMN preads HEADING 'Physical|Reads' FORMAT 9999,999,999;

SELECT substr(to_char(max.run_time_date,'DD-MON-YY'),1,9) "Time",
       max.sid_name "Sid",
       max.redo_log_no - min.redo_log_no "Archive Switches",
       (max.logical_disk_reads_qty-min.logical_disk_reads_qty) lreads,
       (max.physical_disk_reads_qty-min.physical_disk_reads_qty) preads,
       ROUND((((max.logical_disk_reads_qty-min.logical_disk_reads_qty)
              -(max.physical_disk_reads_qty-min.physical_disk_reads_qty))
              /(max.logical_disk_reads_qty-min.logical_disk_reads_qty))*100,3)

```

```

"SGA Hit"
FROM db_level_stats min, db_level_stats max
WHERE max.stats_id_no = &&stats_id_no_max
AND min.stats_id_no = &&stats_id_no_min;

COLUMN database_ver HEADING 'ORACLE Database Version' FORMAT A64;

SELECT oracle_database_version_desc database_ver
FROM db_level_stats
WHERE stats_id_no = &&stats_id_no_max;

set term off;

COLUMN avgusers HEADING 'Average|# Users' FORMAT 990.00;
COLUMN maxusers HEADING 'Maximum|Users' FORMAT 990.00;
COLUMN avgdusers HEADING 'Average|Distinct|# Users' FORMAT 990.00;
COLUMN maxdusers HEADING 'Maximum|Distinct|Users' FORMAT 990.00;
COLUMN avgl15 HEADING 'Average|Load' FORMAT 990.000;
COLUMN maxl1 HEADING 'Maximum|Load' FORMAT 990.000;

set term on;
SELECT AVG(active_total_user_cnt_qty) avgusers,
       Max(active_total_user_cnt_qty) maxusers,
       AVG(active_distinct_user_cnt_qty) avgdusers,
       Max(active_distinct_user_cnt_qty) maxdusers,
       AVG(load_avg_qty_15_min) avgl15,
       MAX(load_avg_qty_1_min) maxl1
FROM db_level_stats db , node_level_stats
WHERE db.stats_id_no <= &&stats_id_no_max
AND db.stats_id_no >= &&stats_id_no_min;

set term on;
SET HEAD off;

SELECT 'Database File Statistics:' FROM dual;
SET HEAD on;

COLUMN totblkswr NEW_VALUE totblkswr HEADING 'Total Blks|Written' FORMAT 999,999
,990;
COLUMN totblkprd NEW_VALUE totblkprd HEADING 'Total Blks|Read' FORMAT 999,999
,990;

SELECT SUM(max.physical_block_reads_qty - min.physical_block_reads_qty)
       totblkprd,
       SUM(max.physical_block_writes_qty - min.physical_block_writes_qty)
       totblkswr
FROM db_file_level_stats min, db_file_level_stats max
WHERE min.oracle_file_id_no = max.oracle_file_id_no
AND min.stats_id_no = &&stats_id_no_min
AND max.stats_id_no = &&stats_id_no_max;

COLUMN name HEADING 'Data File Name' FORMAT a34;
COLUMN file# HEADING 'File#' FORMAT 999;
COLUMN phyblkrd HEADING 'Phy Blks|Reads' FORMAT 99,999,990;
COLUMN bldrdpct HEADING '% Total|Reads' FORMAT 990.00;
COLUMN phyblkwrt HEADING 'Phy Blks|Written' FORMAT 99,999,990;
COLUMN bldwrpct HEADING '% Total|Written' FORMAT 990.00;

```

```

SELECT substr(a.name,1,34) name ,
       a.file# file#,
       (max.physical_block_reads_qty-min.physical_block_reads_qty) phyblkrd,
       ROUND(((max.physical_block_reads_qty-min.physical_block_reads_qty)
              /&&totblksrd)*100,2) bldrdpct,
       (max.physical_block_writes_qty-min.physical_block_writes_qty) phyblkwrt,
       ROUND(((max.physical_block_writes_qty-min.physical_block_writes_qty)
              /&&totblkswr)*100,2) bldwrpct
FROM db_file_level_stats min, db_file_level_stats max, v$dbfile a
WHERE a.file# = min.oracle_file_id_no
AND a.file# = max.oracle_file_id_no
AND max.stats_id_no = &&stats_id_no_max
AND min.stats_id_no = &&stats_id_no_min
order by (max.physical_block_reads_qty-min.physical_block_reads_qty) DESC;

```

```

COLUMN disk          HEADING 'Disk'  FORMAT a15;
COLUMN phyblksum     HEADING 'Total|Phy Bks' FORMAT 99,999,990;
COLUMN bldsumpct     HEADING '% of|Total' FORMAT 990.00;

```

```

SELECT substr(a.name,1,15) disk ,
       SUM(max.physical_block_reads_qty-min.physical_block_reads_qty) phyblkrd,
       ROUND((SUM(max.physical_block_reads_qty-min.physical_block_reads_qty)
              /&&totblksrd)*100,2) bldrdpct,
       SUM(max.physical_block_writes_qty-min.physical_block_writes_qty)
       phyblkwrt,
       ROUND((SUM(max.physical_block_writes_qty-min.physical_block_writes_qty)
              /&&totblkswr)*100,2) bldwrpct,
       SUM(max.physical_block_reads_qty-min.physical_block_reads_qty)
       +SUM(max.physical_block_writes_qty-min.physical_block_writes_qty)
       phyblksum,
       ROUND((SUM((max.physical_block_reads_qty-min.physical_block_reads_qty)
                  +(max.physical_block_writes_qty-min.physical_block_writes_qty))
              /(&&totblksrd+&&totblkswr))*100,2) bldsumpct
FROM db_file_level_stats min, db_file_level_stats max, V$DBFILE a
WHERE a.file# = min.oracle_file_id_no
AND a.file# = max.oracle_file_id_no
AND max.stats_id_no = &&stats_id_no_max
AND min.stats_id_no = &&stats_id_no_min
group by substr(a.name,1,15);

```

```

spool off;

```

Appendix B: Sample Report Output:

"Database Performance Report"						
Time	Sid	Archive Switches	Logical Reads	Physical Reads	SGA Hit	
10-JUL-95	AISP2	21	38,753,147	727,473	98.123	
ORACLE Database Version						
Oracle7 Server Release 7.1.4.1.10 - Production Release						
Average Maximum						
Average # Users	Maximum # Users	Distinct # Users	Distinct Users	Average Load	Maximum Load	
34.25	58.00	5.63	15.00	0.345	1.290	
Database File Statistics:						
Total Blks Read	Total Blks Written					
727,473	444,058					
Data File Name	Phy File#	Blks Reads	% Total Reads	Phy Blks Written	% Total Written	
/ora_db2z/aisp2/fineftab01.dbf	8	227,201	31.23	2,319	0.52	
/ora_db2f/aisp2/idxctab01.dbf	13	143,000	19.66	170,918	38.49	
/ora_db2e/aisp2/idxbtab01.dbf	14	85,636	11.77	88,090	19.84	
/ora_db2z/aisp2/finaptab01.dbf	16	72,467	9.96	62	0.01	
/ora_db2d/aisp2/idxatab01.dbf	11	63,735	8.76	78,096	17.59	
/ora_db2n/aisp2/finapexatab01.dbf	12	29,595	4.07	21,686	4.88	
/ora_db2y/aisp2/finapexatab01.dbf	17	27,963	3.84	18	0.00	
/ora_db2y/aisp2/finglxtab01.dbf	27	26,969	3.71	630	0.14	
/ora_db2z/aisp2/fingltab01.dbf	15	19,455	2.67	55	0.01	
/ora_db2y/aisp2/lininsttab01.dbf	9	9,487	1.30	2,777	0.63	
/ora_db2z/aisp2/linlaftab01.dbf	10	5,834	0.80	829	0.19	
/ora_db2a/aisp2/system01.dbf	1	4,143	0.57	387	0.09	
/ora_db2z/aisp2/linlaftab02.dbf	32	3,560	0.49	1,902	0.43	
/ora_db2n/aisp2/lptab01.dbf	20	2,600	0.36	0	0.00	
/ora_db2z/aisp2/finaptab02.dbf	31	2,469	0.34	438	0.10	
/ora_db2w/aisp2/boppotab01.dbf	18	1,709	0.23	142	0.03	
/ora_db2c/aisp2/sysrbb01.dbs	25	470	0.06	39,410	8.87	
/ora_db2z/aisp2/fingltab02.dbf	40	439	0.06	212	0.05	
/ora_db2b/aisp2/sysrba01.dbs	26	348	0.05	33,348	7.51	
/ora_db2a/aisp2/users01.dbf	5	183	0.03	48	0.01	
/ora_db2z/aisp2/fndshrtab01.dbf	19	88	0.01	0	0.00	
/ora_db2a/aisp2/systools01.dbf	7	85	0.01	2	0.00	
/ora_db2y/aisp2/bgtab01.dbf	24	20	0.00	39	0.01	
/ora_db2y/aisp2/misctab01.dbf	28	14	0.00	0	0.00	
/ora_db2z/aisp2/fndshrtab03.dbf	39	3	0.00	0	0.00	
/ora_db2a/aisp2/syssort01.dbf	2	0	0.00	2,650	0.60	
/ora_db2n/aisp2/tmptab01.dbf	6	0	0.00	0	0.00	

